Exception Handling

What is an exception?

**Exception** is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program’s instructions, and terminate the program abruptly.

What is Exception handling?

Exception handling is a way to handle the runtime errors so that regular flow of program can be preserved.

Class Test{

P s v m(String[] args){

Sopln(“1”);

-//-

Sopln(“9”);

Sopln(100/0);

Sopln(“10”);

-//-

Sopln(“20”);

}

}

o/p 1 to 9 and then Exception ArithmaticException and termination of program.

Exception Hierarchy



Difference between Exception and Error

|  |  |
| --- | --- |
| Exception | Error |
| Exception occur because of our program | Error occur because of lack of system resources |
| Exception are recoverable I’e programmer can handle them using try-catch block | Error are not recoverable by programmer it handle by system admin |
| Type of exception  1)compiled time or checked exception  2) rum time or unchecked exception | Types of error  Runtime |

**Error:**An Error indicates a serious problem that a reasonable application should not try to catch.

**Exception:**Exception indicates conditions that a reasonable application might try to catch.

Note: all exceptions are occur at runtime not at compile time.

import java.io.FileInputStream;

class Teat{

p s v m(String[] args){

FileInputStream fis=new FileInputStream(“d:/abc.txt”);

Class.forName(“com.mysql.jdbc.Driver”);

int a=100, b=0, c;

c=a/b;

sopln(c);

}

}

Checked Exception

Theseare the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the [*throws*keyword](https://www.geeksforgeeks.org/throw-throws-java/). In checked exception, there are two types: fully checked and partially checked exceptions. A fully checked exception is a checked exception where all its child classes are also checked, like IOException, InterruptedException. A partially checked exception is a checked exception where some of its child classes are unchecked, like Exception.

import java.io.\*;

// Main class

class GFG {

    // Main driver method

    public static void main(String[] args)

    {

        // Reading file from path in local directory

        FileReader file = new FileReader("C:\\test\\a.txt");

        // Creating object as one of ways of taking input

        BufferedReader fileInput = new BufferedReader(file);

        // Printing first 3 lines of file "C:\test\a.txt"

        for (int counter = 0; counter < 3; counter++)

            System.out.println(fileInput.readLine());

        // Closing file connections

        // using close() method

        fileInput.close();

    }

}

This program give compile time exception which can be handle by using throws or try-catch block

import java.io.\*;

// Main class

class GFG {

    // Main driver method

    public static void main(String[] args)

        throws IOException

    {

        // Creating a file and reading from local repository

        FileReader file = new FileReader("C:\\test\\a.txt");

        // Reading content inside a file

        BufferedReader fileInput = new BufferedReader(file);

        // Printing first 3 lines of file "C:\test\a.txt"

        for (int counter = 0; counter < 3; counter++)

            System.out.println(fileInput.readLine());

        // Closing all file connections

        // using close() method

        // Good practice to avoid any memory leakage

        fileInput.close();

    }

}

Unchecked Exception

These are the exceptions that are not checked at compile time. In Java, exceptions under *Error*and *RuntimeException*classes are unchecked exceptions.

class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Here we are dividing by 0

        // which will not be caught at compile time

        // as there is no mistake but caught at runtime

        // because it is mathematically incorrect

        int x = 0;

        int y = 10;

        int z = y / x;

    }

}

Above program compile fine but give exception at the runtime.

Difference between checked and unchecked exception

|  |  |
| --- | --- |
| Checked/Compile time Exception | Unchecked/runtime Exception |
| Checked exception are the exception which are handle at the compile time | Unchecked exception are the exception that are not able to checked at compiled time |
| Program gives compilation error if method throws a checked exception | Program compile fine because compiler is not able to check the exception |
| If some code within the method throws a checked exception then the method must specify the exception using throws keyword | Method is not forced by compiler to declare the unchecked exception thrown by its implementation generally, such methods almost do not declare them. |
| Checked exception are occur when chances of failure are too high | Unchecked exception are occur mostly due to programming mistake. |
| They are direct subclass of exception class and extends Exception class | Direct subclass of RuntimeException and extends RunTimeException class. |

What happened when an Exception occur/ How JVM handle exception

Whenever an exception occur at particular line then method associated with that line will create object that object is called exception object.

JVM asked to main method have you handle exception if no then JVM pass exception to Default Exception Handler

And it print error and terminate main method abnormally.

Exception Object

Note:

There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of the methods is called **Call Stack**

* The run-time system searches the call stack to find the method that contains a block of code that can handle the occurred exception. The block of the code is called an **Exception handler**.
* The run-time system starts searching from the method in which the exception occurred, and proceeds through the call stack in the reverse order in which methods were called.
* If it finds an appropriate handler, then it passes the occurred exception to it. An appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
* If the run-time system searches all the methods on the call stack and couldn’t have found the appropriate handler, then the run-time system handover the Exception Object to the **default exception handler**, which is part of the run-time system. This handler prints the exception information in the following format and terminates the program **abnormally**.

class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Taking an empty string

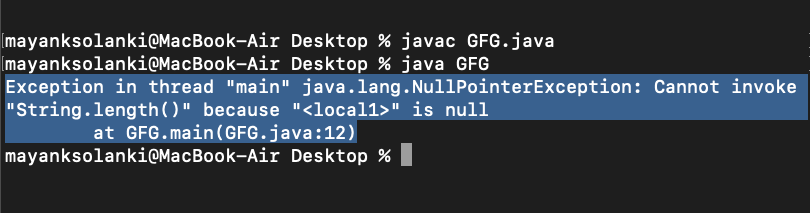
        String str = null;

        // Getting length of a string

        System.out.println(str.length());

    }

}



Methods to print Exception Information

1. e.printStackTrace()

use to print exception name, description and stack trace

class GFG {

    public static void main (String[] args) {

      int a=5;

      int b=0;

        try{

          System.out.println(a/b);

        }

      catch(ArithmeticException e){

        e.printStackTrace();

      }

    }

}

1. e.toString()

use to print name of the exception

class GFG1 {

    public static void main (String[] args) {

      int a=5;

      int b=0;

        try{

          System.out.println(a/b);

        }

      catch(ArithmeticException e){

        System.out.println(e.toString());

      }

    }

}

1. e.getMessage()

class GFG1 {

public static void main (String[] args) {

int a=5;

int b=0;

try{

System.out.println(a/b);

}

catch(ArithmeticException e){

System.out.println(e.getMessage());

}

}

}

How Programmer handle an Exception

Java exception handle by using below keywords

1. try
2. catch
3. throw
4. throws
5. finally

try-catch

try

{

//risky code

}

catch(ExceptionClassName ref.var.name)

{

//handling code

}

import java.io.InputStream

class Test{

p s v m(String[] args){

sopln(“1”);

try

{

Sopln(“2”);

int a=100; b=0; c;

c=a/b;

sopln(“4”);

sopln(c);

sopln(“5”);

}

catch(Exception e)

{

sopln(“6”);

sopln(e);

sopln(‘7”);

}

sopln(“Hello”);

}

}

Note: catch block will only execute if exception occur unless not, if exception occur the code outside catch block will not execute.

finally block

it is the block which always execute, we write clean up code in this block

eg: db connection

class Test{

p s v m(String[] args){

try

{

int a=100, b=0, c;

c=a/b;

sopln(c);

}

catch(ArithmaticException e)

{

sopln(“Exception” +e);

}

finally

{

sopln(“in finally block”);

}

}

}

try

{

//risky code, file open/close, read write

}

catch(Exception e)

{

//handling code

}

finally

{

//Clean up code, always executable code, closing DB connection

}

Different ways of writing try, catch and finally

1)

try/catch/finally

{

}

Not possible

2)

try

{

}

catch(Exception e)

{

}

3)

try

{

}

catch(ArithmaticException e) // child exception

{

}

catch(Exception e) // parent exception

{

}

4)

try

{

}

catch(Exception e)

{

}

finally

{

}

5)

try

{

}

finally

{

}

6)

Try

{

}

finally

{

}

try

{

}

catch(Exception e)

{

}

6)nested try catch

Try

{

Try

{

}

catch(Exception e)

{

}

}

catch(Exception e)

}

Difference between final, finally and finaliser method

Final is a keyword

Finally is a block

Finaliser is a method which is executed just before garbage collector, the object which are going to delete the resources related to that objects are closed inside finalize method

Executed just before GC.

throw keyword

java throw keyword is use to throw exception explicitly, it create exception object and handover that object to JVM

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.

We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number. Here, we just need to set the condition and throw exception using throw keyword.

Syntax

1. **throw** **new** exception\_class("error message");

class Test{

p s v m(String[] args){

int a=100, b=0, c;

c=a/b;{

throw new ArithmaticException();

}

}

}

Note: here throw keyword create exception object and that object pass to JVM and default exception handler print error

public class TestThrowException extends RuntimeException {

//function to check if person is eligible to vote or not

public static void validate(int age) {

if(age<18) {

//throw Arithmetic exception if not eligible to vote

throw new ArithmeticException("Person is not eligible to vote"); //line 1

}

else {

System.out.println("Person is eligible to vote!!");

Sopln(“Hello”) //not allowed

}

}

//main method

public static void main(String args[]){

//calling the function

validate(13);

System.out.println("rest of the code...");

}

}

Note: in above eg line 1 create explicit exception and handle to JVM and default exception handler print error

We cannot write anything after throw statement

Class YongerAgeException extends RuntimeException{

public static void validate(int age) {

try

{

if(age<18)

{

//throw Arithmetic exception if not eligible to vote

throw new YongerAgeException ("Person is not eligible to vote"); //line 1

}

else

{

System.out.println("Person is eligible to vote!!");

Sopln(“Hello”) //not allowed

}

}

catch(YongerAgeException e)

{

e.printStackTrace(e);

}

}

//main method

public static void main(String args[]){

//calling the function

validate(13);

System.out.println("rest of the code...");

}

}

Note:

throw create exception object manually(by programmer) and handover to JVM

we can throw either checked or unchecked exception but throw is best for customized exception

we can throw class that comes in throwable child class

we cannot write any statement after throw, otherwise it will provide unreachable statement error.

throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

The exception which is thrown by the method should be handle by the method which call that method.

Throws keyword is only used with checked exception

1. return\_type method\_name() **throws** exception\_class\_name{
2. //method code
3. }

Note: throws’ keyword is used to declare an exception, it gives an information to the caller method that there may occur an exception so it is better for the caller method to provide the exception handling code so that the normal flow can be maintained.

import java.io.FileInputStream;

import java.io.FileNotFoundExeption;

import java.io.FileOutputStream;

class readAndWrite{

void readFile() throws FileNotFoundExeption{

FileInputStream fis=new FileInputStream(“d/abc.txt”);

//statement

}

void saveFile() throws FileNotFoundExeption{

String text=”this is done”;

FileOutputStream fos=new FileOutputStream(“d/xyz.txt”);

//statement

}

}

class Test{

p s v m(String[] args){

readAndWrite rw=new readAndWrite();

try

{

rw.readFile();

}

catch(FileNotFoundExeption e)

{

e.printStackTrace();

}

try

{

rw.saveFile();

}

catch(FileNotFoundExeption e)

{

e.printStackTrace();

}

Note: throws keyword is not recommended, it is just to convinced the compiler

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

class tst

{

    public static void main(String[] args)

    {

        Thread.sleep(10000);

        System.out.println("Hello Geeks");

    }

}

error: unreported exception InterruptedException; must be caught or declared to be thrown

In the above program, we are getting compile time error because there is a chance of exception if the main thread is going to sleep, other threads get the chance to execute main() method which will cause InterruptedException.

|  |
| --- |
| class tst  {      public static void main(String[] args)throws InterruptedException      {          Thread.sleep(10000);          System.out.println("Hello Geeks");      }  } |

Hello Geeks

In the above program, by using throws keyword we handled the InterruptedException and we will get the output as **Hello Geeks**

class ThrowsExecp

{

    static void fun() throws IllegalAccessException

    {

        System.out.println("Inside fun(). ");

        throw new IllegalAccessException("demo");

    }

    public static void main(String args[])

    {

        try

        {

            fun();

        }

        catch(IllegalAccessException e)

        {

            System.out.println("caught in main.");

        }

    }

}

* throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
* throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
* By the help of throws keyword we can provide information to the caller of the method about the exception.

Difference between throw and throws keyword

|  |  |
| --- | --- |
| throw keyword | throws keyword |
| throw keyword is used to create an exception object manually I’e by programmer | throws keyword is used to declare the exception I’e it indicate the caller method that given type of exception may occur so you have to handle it while calling |
| It is mainly used for runtime exception or unchecked exceptions | Throws keyword is mainly used for compile time exception or checked exceptions |
| In case of throw keyword we can only throw only single exception | In case of throws keyword we can declare multiple exception |
| throw keyword is used within the method | throws keyword is used with method signature |
| It followed by new instance | throws keyword is followed by class |
| We cannot write any statement after throw keyword and thus it can be used to break the statement | Throws keyword does not have any such code |

User defined exception

class MyException extends Exception {

    public MyException(String s)

    {

        // Call constructor of parent Exception

        super(s);

    }

}

// A Class that uses above MyException

public class Main {

    // Driver Program

    public static void main(String args[])

    {

        try {

            // Throw an object of user defined exception

            throw new MyException("GeeksGeeks");

        }

        catch (MyException ex) {

            System.out.println("Caught");

            // Print the message from MyException object

            System.out.println(ex.getMessage());

        }

    }

}